# Messaging

## in distributed systems

micro-23

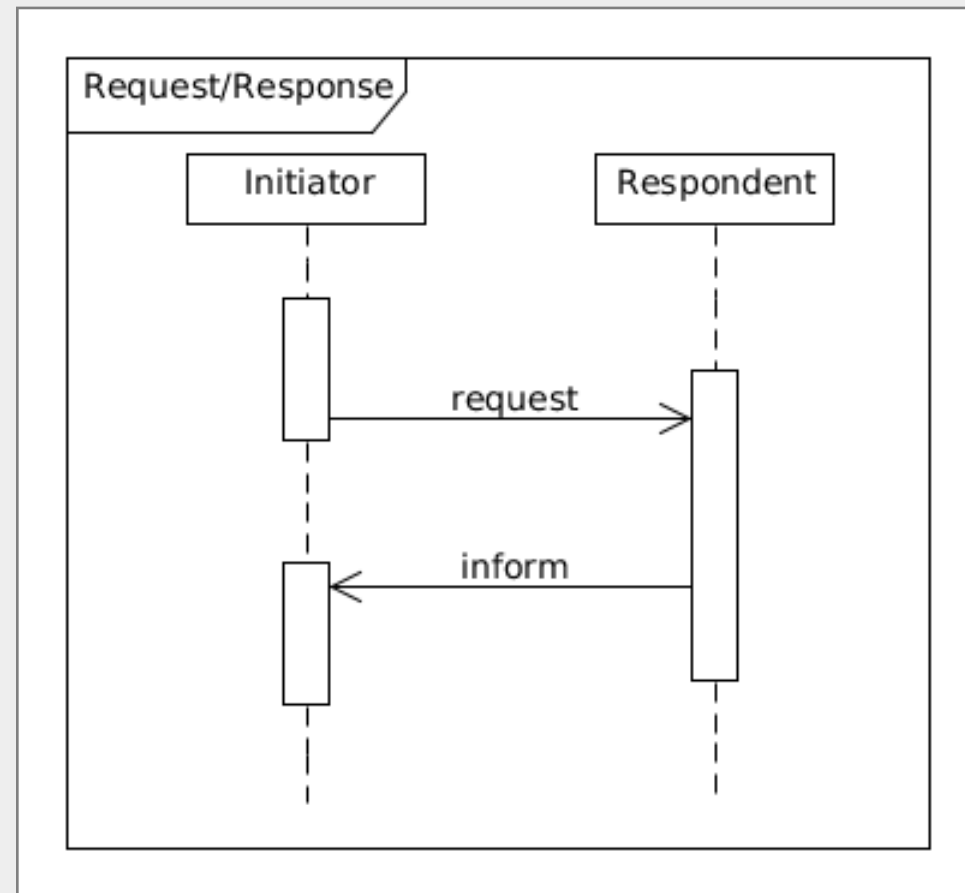Aliaksei Bialiauski

Designed in LaTeX

Request-Response vs Producer-Consumer

Messaging Patterns

Internals of messaging

Chapter #1:

## Request-Response vs Producer-Consumer

Request-Response

[ Availability Performance Backpressure Retry Async ]

## Availability

What should the client do when the server is not available?

[ Availability Performance Backpressure Retry Async ]

## Performance

What should the client do when the server gets slower?
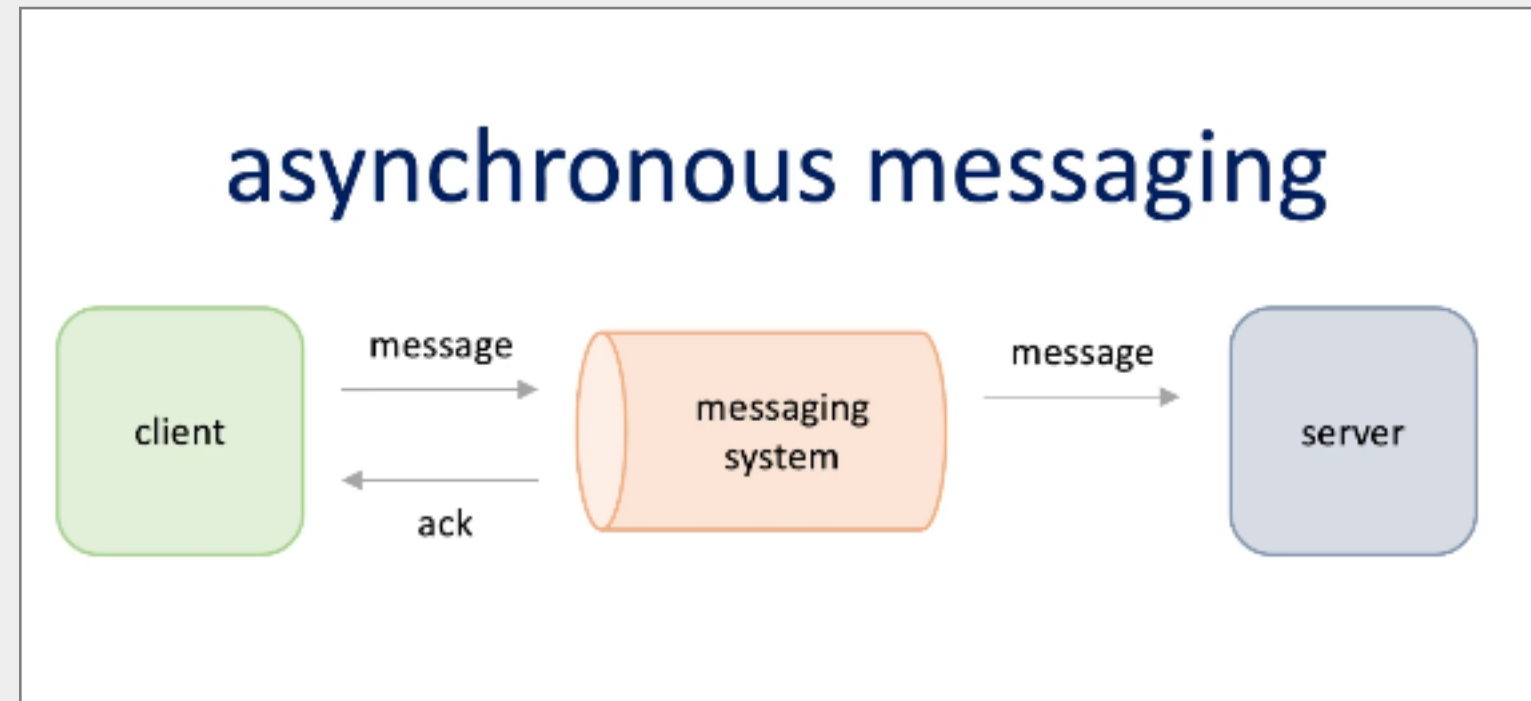
[ Availability Performance Backpressure Retry Async ]

## Backpressure

What should the server do when there is a sudden traffic spike?

## Retry

What should the client do with failed requests?

[ Availability Performance Backpressure Retry Async ]

## Asynchronous messaging

Availability -> Keep messages and send them later

Performance -> Add more servers to parallelize processing

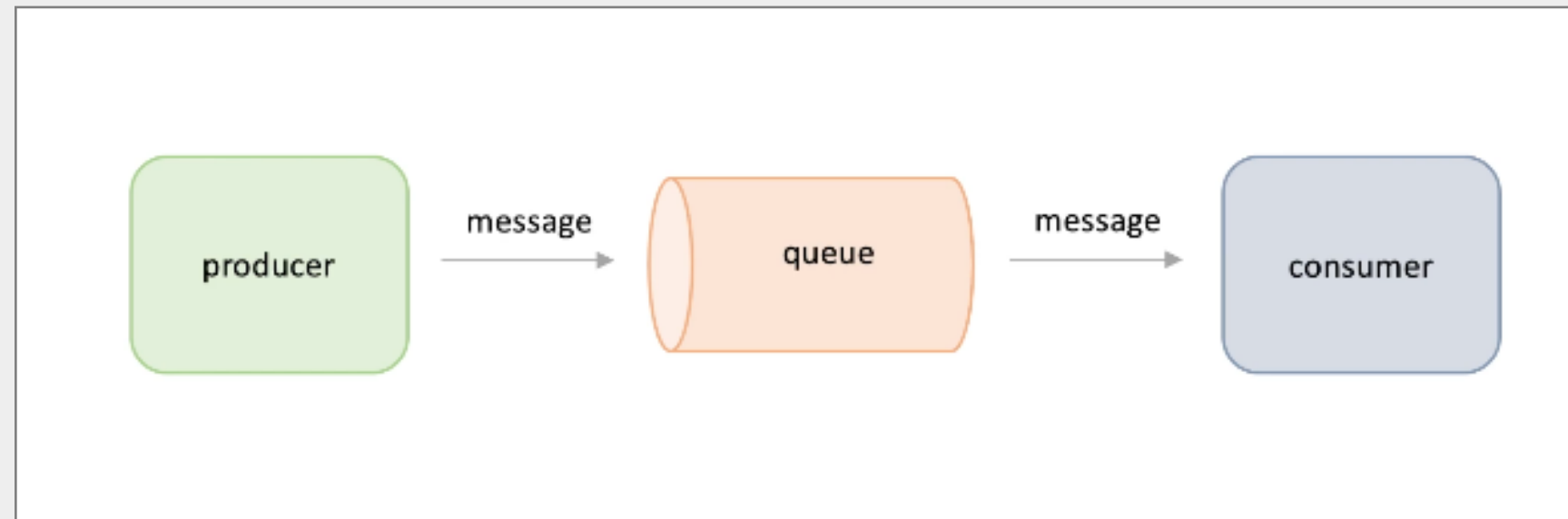Backpressure -> Keep draining messages at its own pace

Retry -> Re-send failed messages

Also, asynchronous messaging helps to decouple client and server.
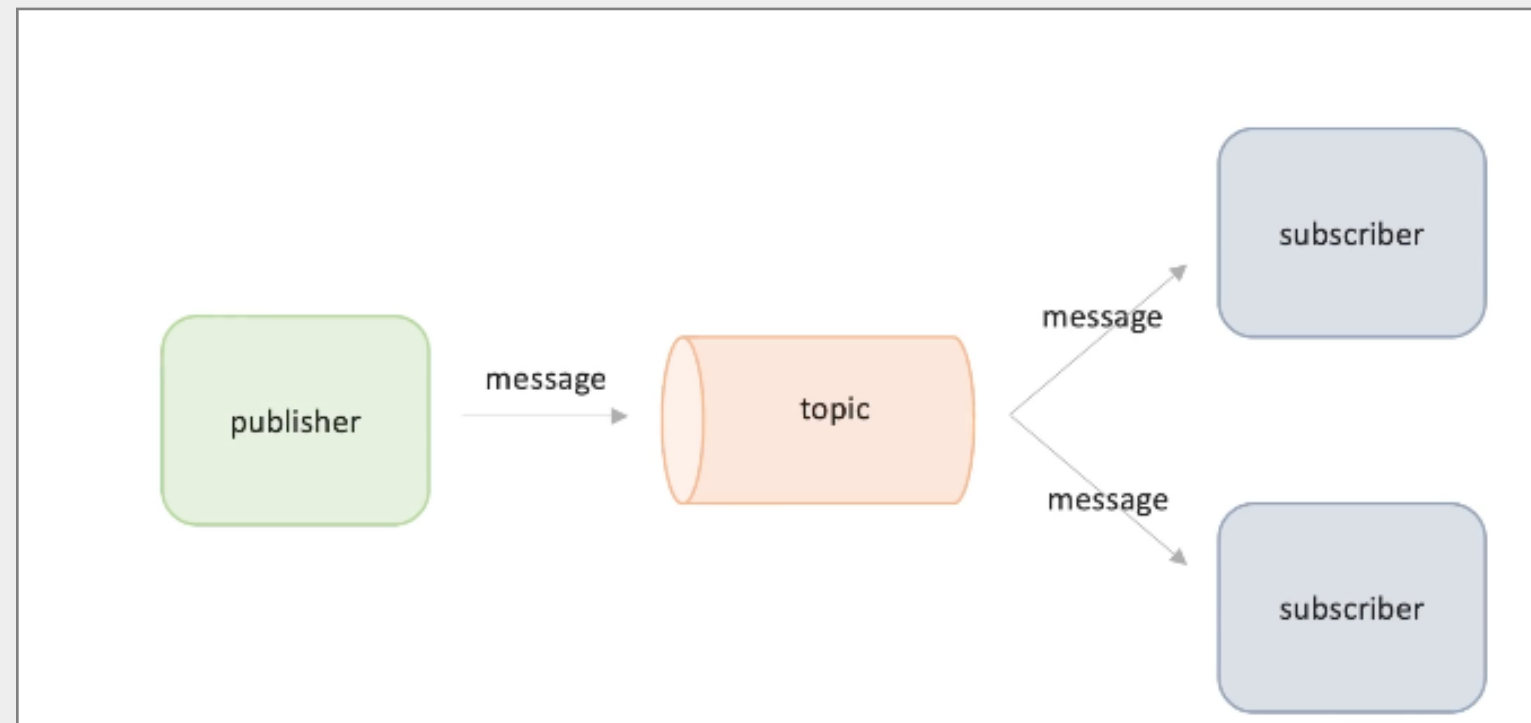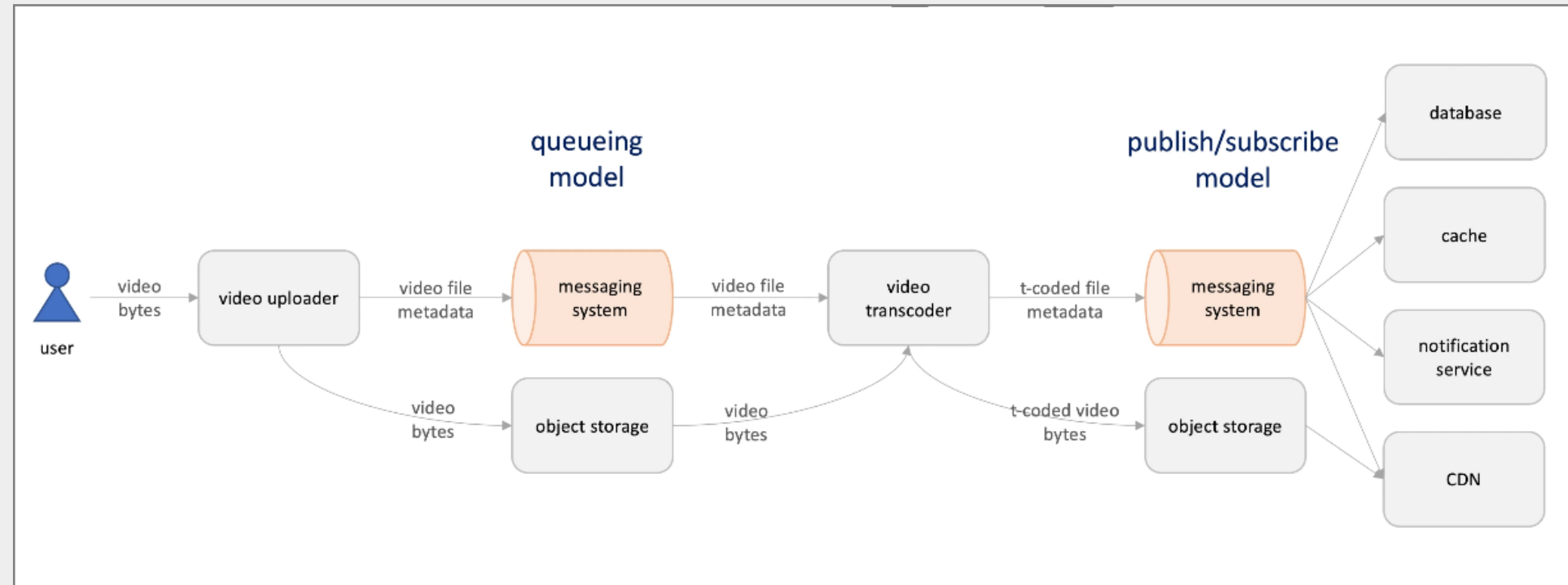
Chapter #2:
Messaging Patterns

## Message Queuing



https://aws.amazon.com/sqs
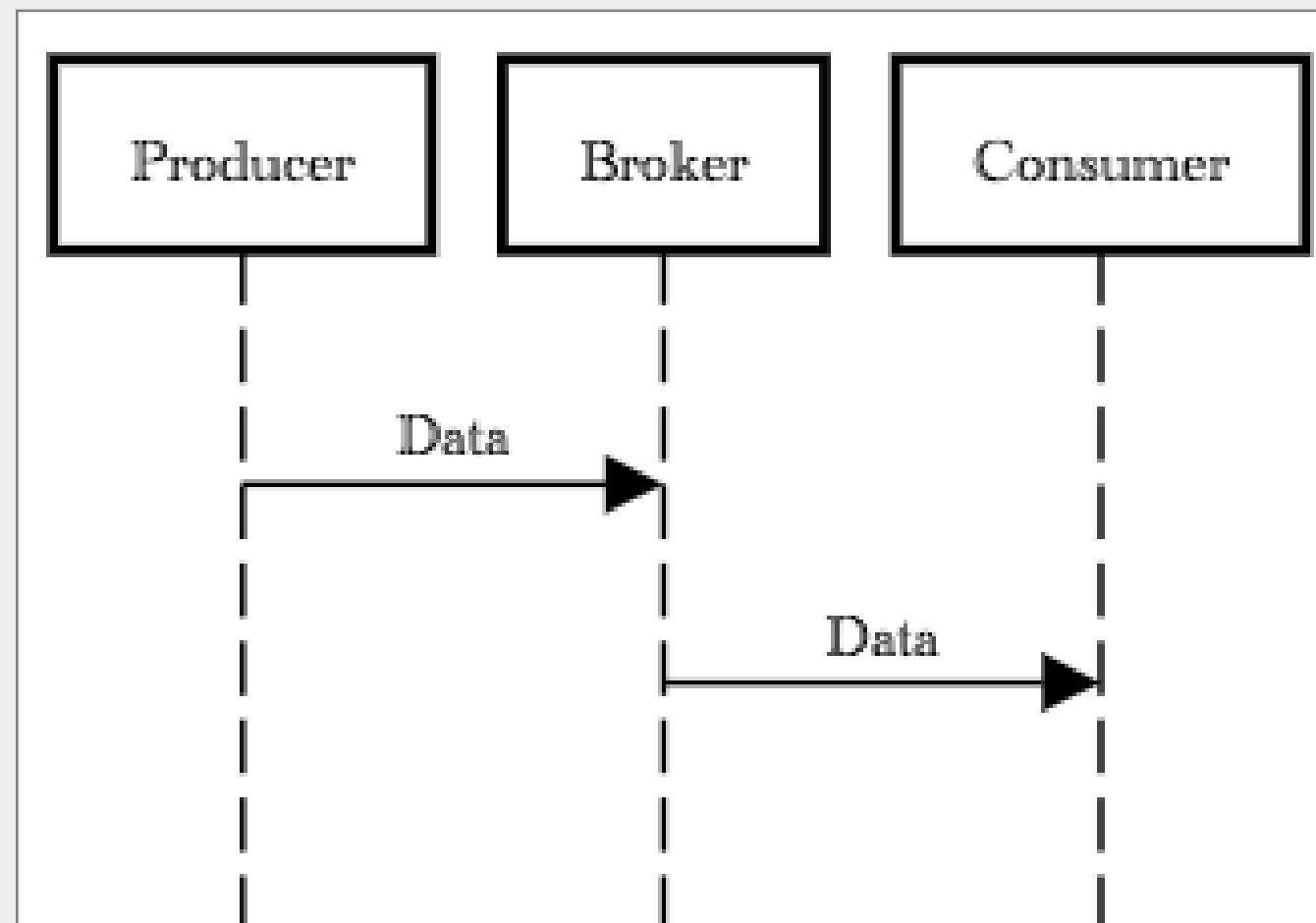
https://www.rabbitmq.com

## Publish-Subscribe



https://kafka.apache.org

https://aws.amazon.com/kinesis

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]
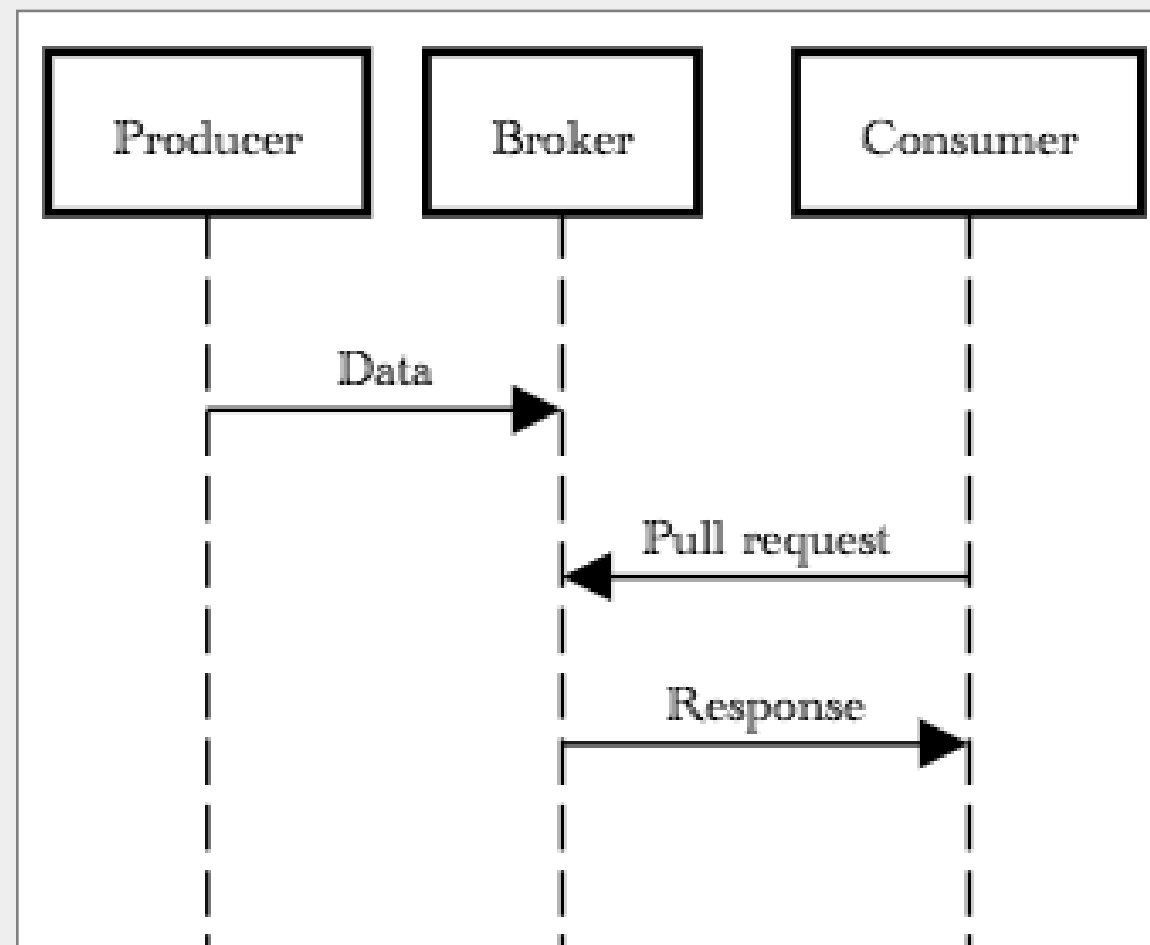
[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]

## Push model

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]

## Pull model

Producer       Broker       Consumer

Data →

← Pull request

Response →

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]

## Push vs Pull

**push**

pros

- optimized for latency
- client (consumer) is lightweight
- load balancing among competing consumers

cons

- flow control
- broker logic is more complex

**push-based**

RabbitMQ, ActiveMQ

**pull**

pros

- optimized for throughput
- better suited for message replay

cons

- polling in a tight loop
- client (consumer) logic is more complex

**pull-based**

Kafka, SQS

## Competing Consumers



**scalability**
add more instances as the number
of messages increases over time

**availability**
when some instances fail,
others continue to process messages

**performance**
process more messages in parallel
by adding more instances

## Claim Check

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]

When to retry?

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]

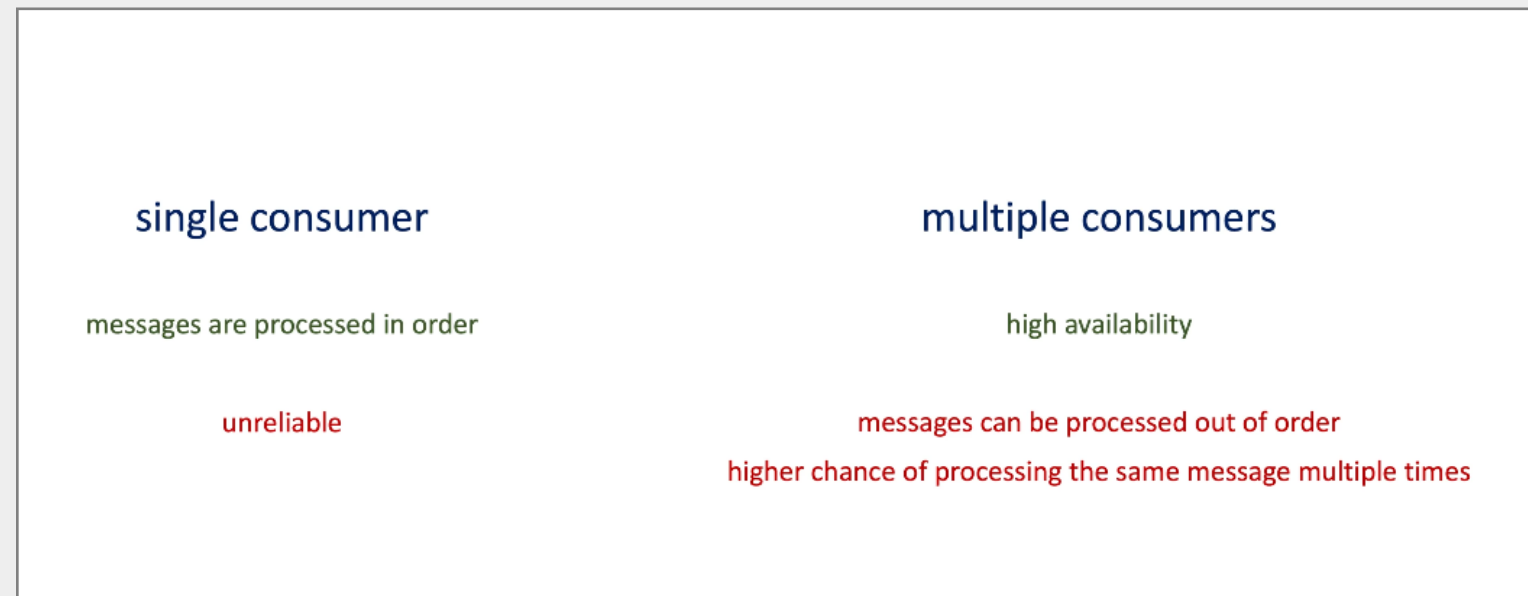| HTTP Status Code | Description | Retry? | Comments |
|---|---|---|---|
| N/A | Network connection is not established. | Yes | It doesn't matter if the API is idempotent or not. If no connection was established, the request was never sent. |
| 400 | Bad request. | No | There is a problem with the request. For example, a required parameter is not supplied for the specified action. Retry will not help. |
| 403 | Access denied. | No | One of many possible reasons - the provided security credentials are not valid. |
| 404 | Not found exception. | No | There is no requested resource present on the server side. E.g. the specified S3 bucket does not exist. |
| 429 | Too many requests. | Yes (but after a delay) | The request was not processed. It was immediately rejected by the server due to the high request rate. |
| 500 | Internal server error. | Yes (if idempotent) | The service encountered an unexpected condition that prevented it from fulfilling the request. |
| 503 | Service unavailable. | Yes | The requested service is not available. |

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]

**Exponential backoff**

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]
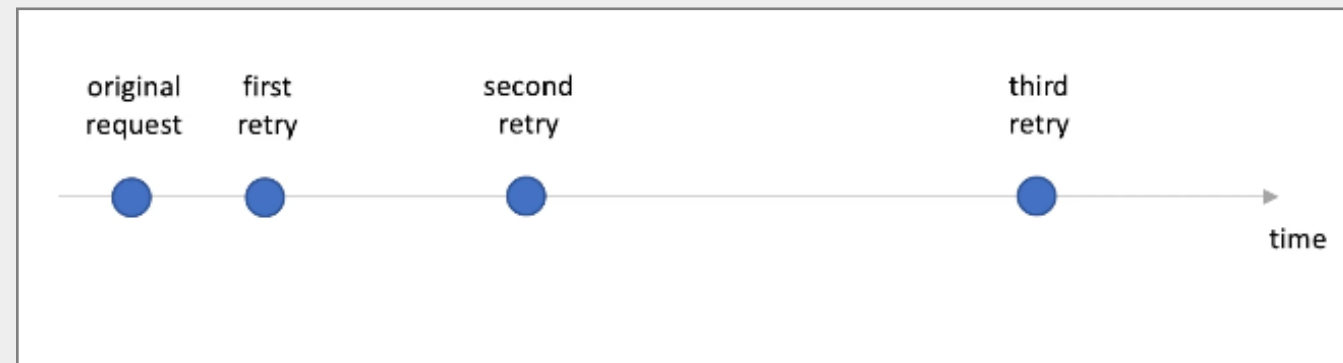
Retry at different times
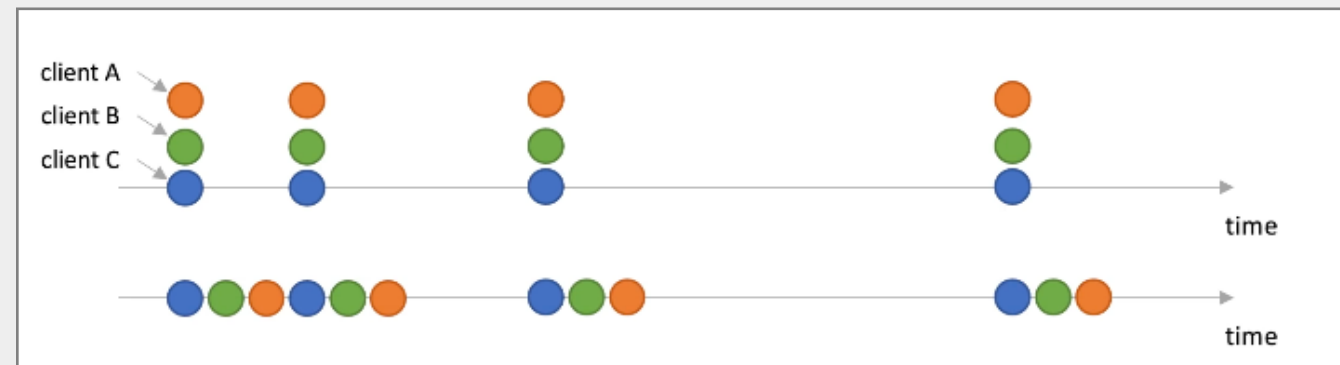
## Sigle consumer vs Multiple consumers

single consumer

messages are processed in order

unreliable

multiple consumers

high availability

messages can be processed out of order

higher chance of processing the same message multiple times

Multiple consumers break the order of messages

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]
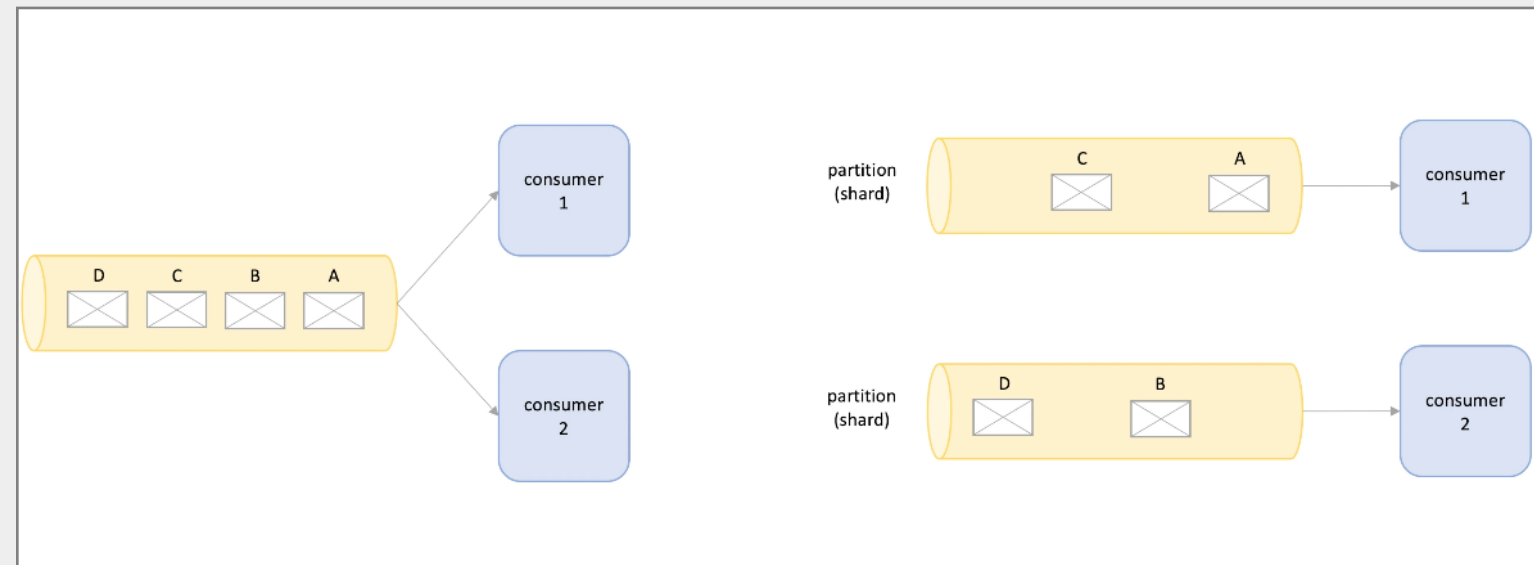
## Data partitioning



Types: Lookup, Range, Hash

[ MQ Pub-Sub P&P Competing Claim Retry Partitioning ]

## Netflix architecture
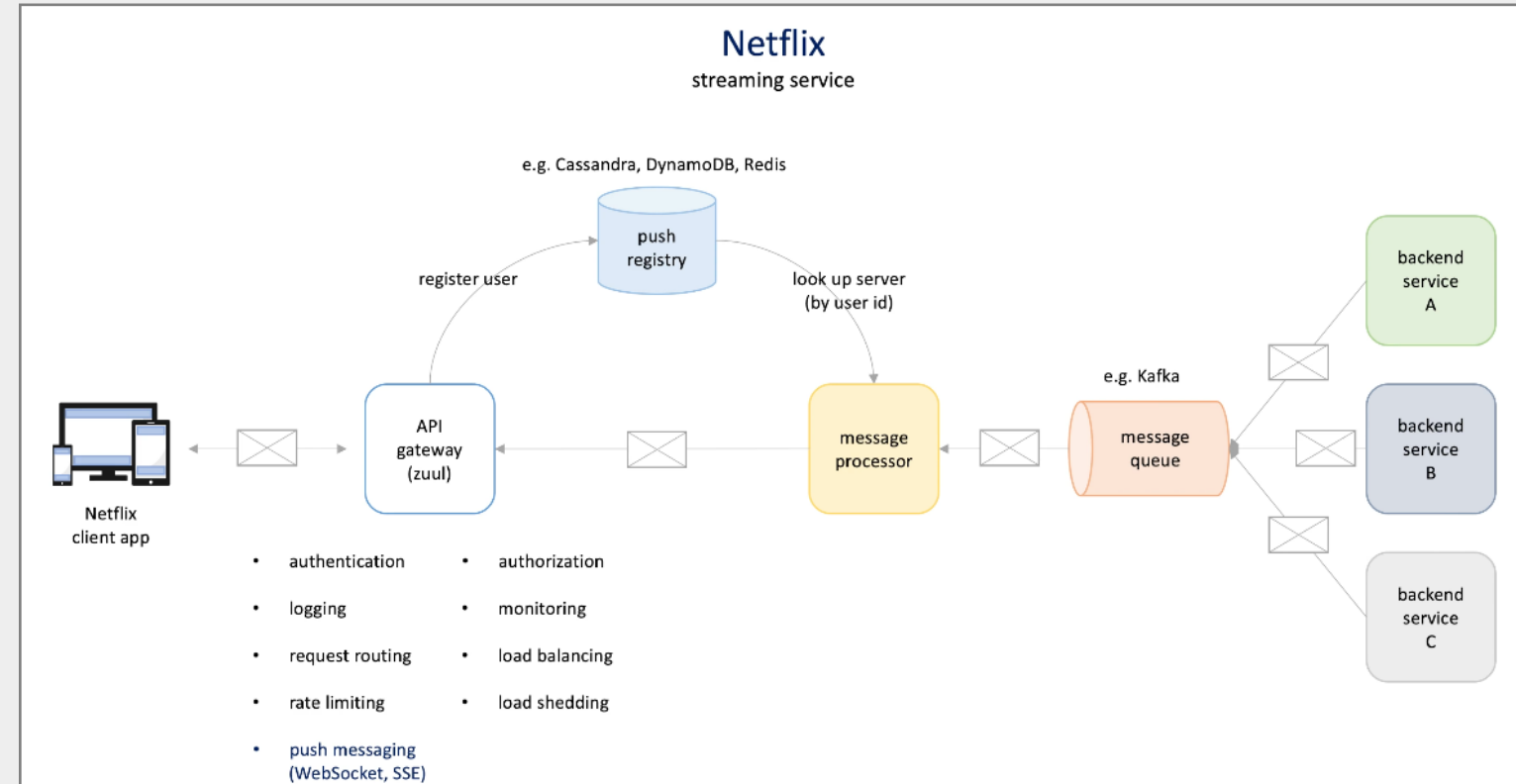
Chapter #3:
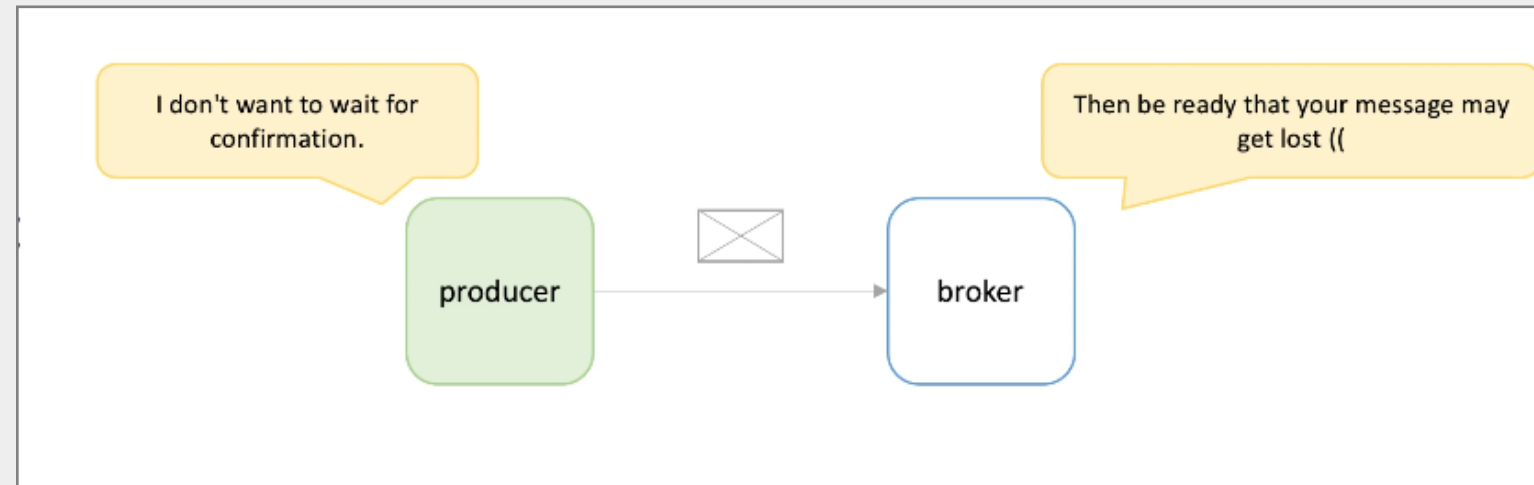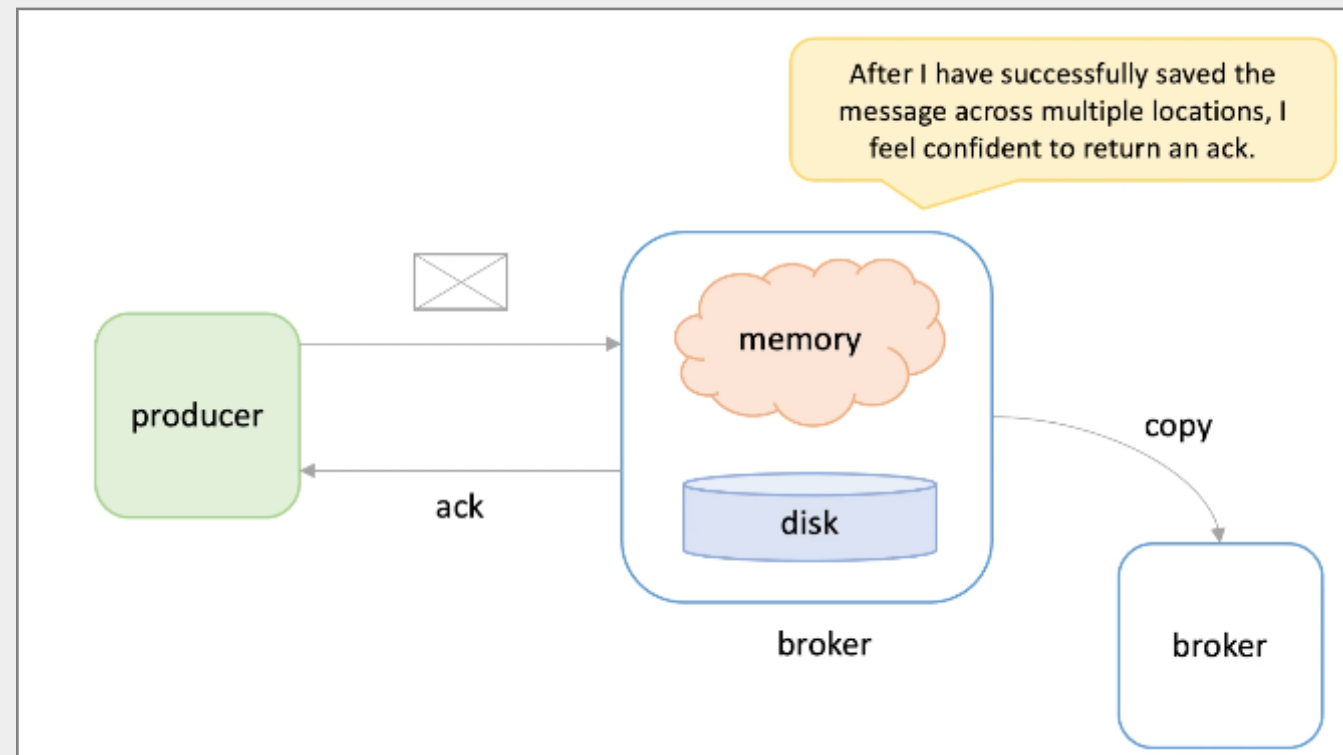Internals of messaging

[ Ack Offset Polling Guarantees ]

## Message Acknowledgement

[ Ack Offset Polling Guarantees ]
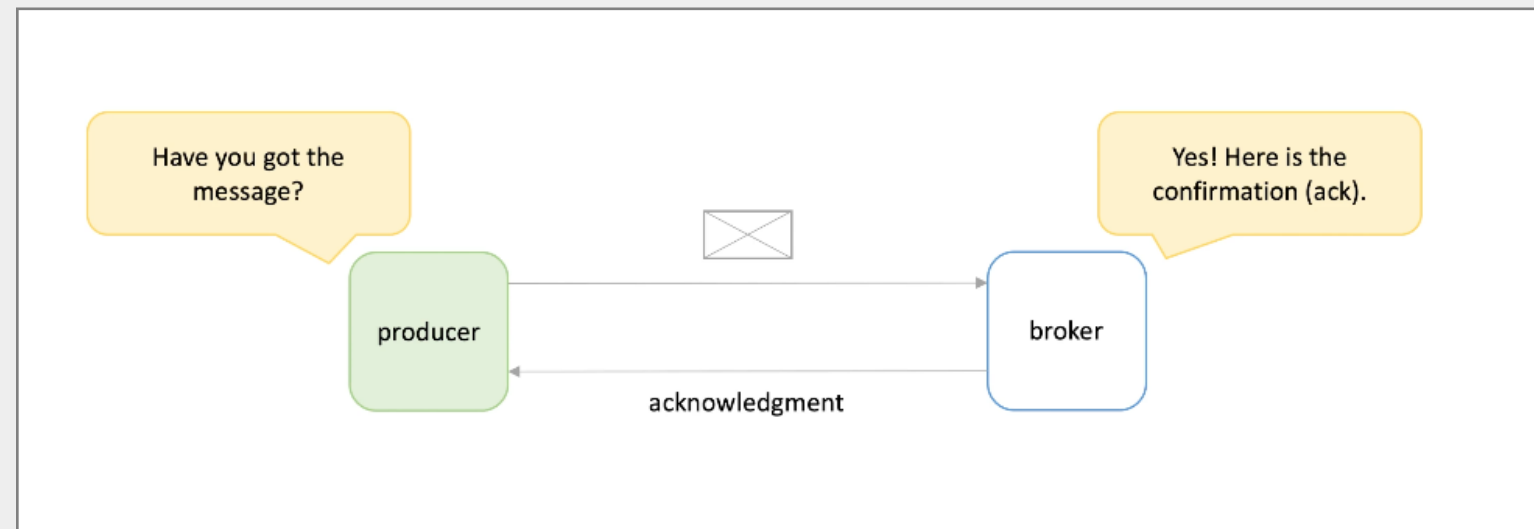
## Unsafe ack aka "fire and forget"

[ Ack Offset Polling Guarantees ]

Safe ack



It is up to the broker what safe really means

[ Ack Offset Polling Guarantees ]

Failed ack
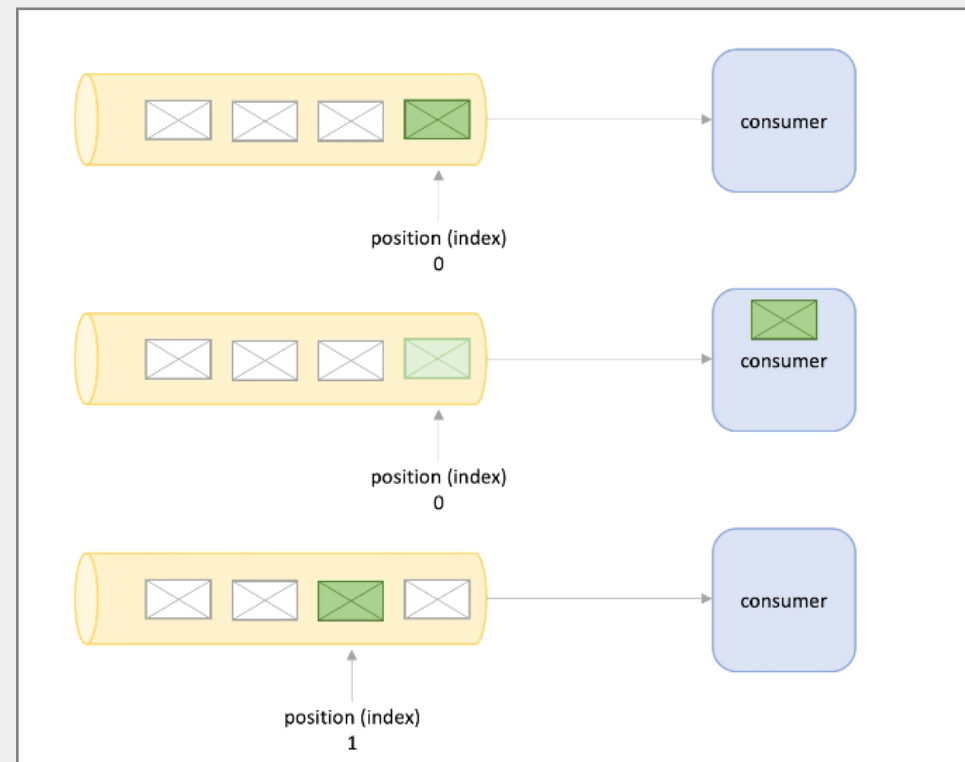
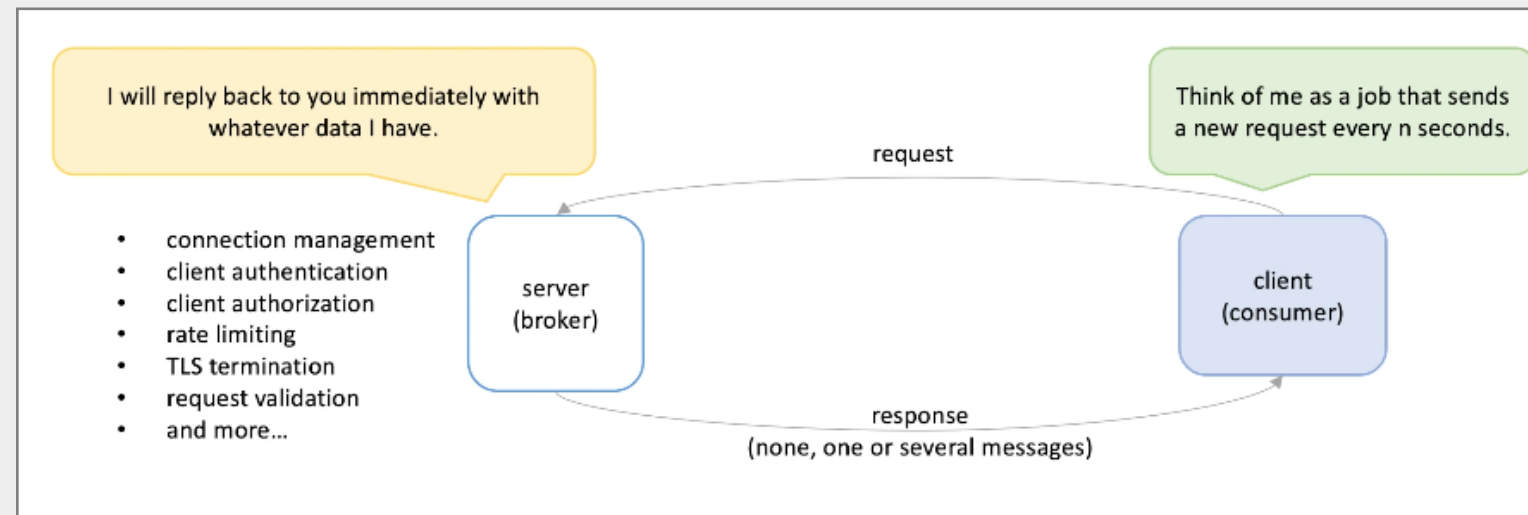[ Ack Offset Polling Guarantees ]
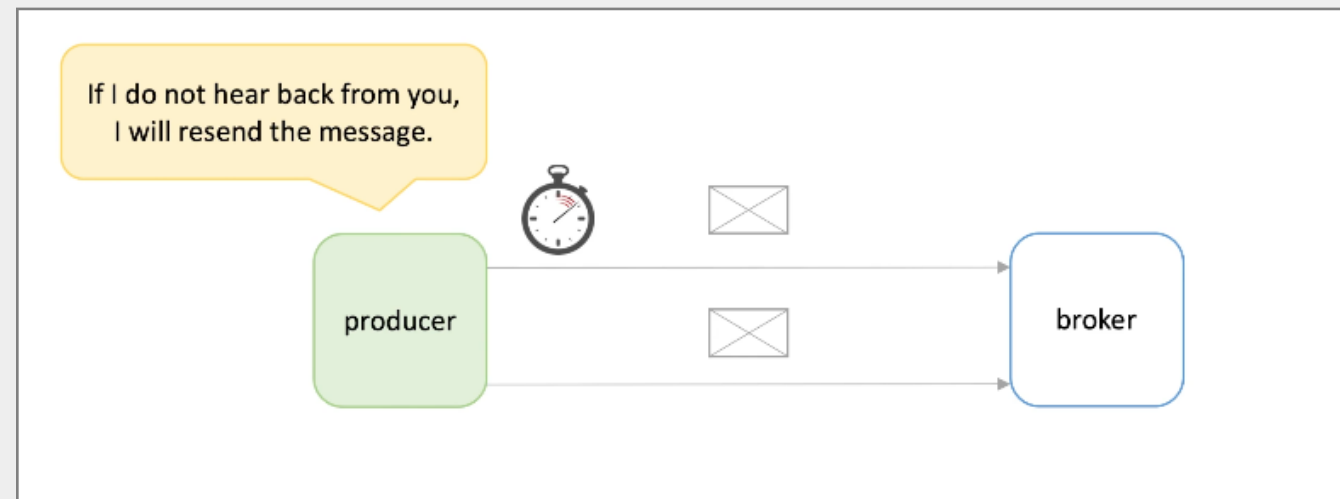
## Consumer Offsets - Head delete
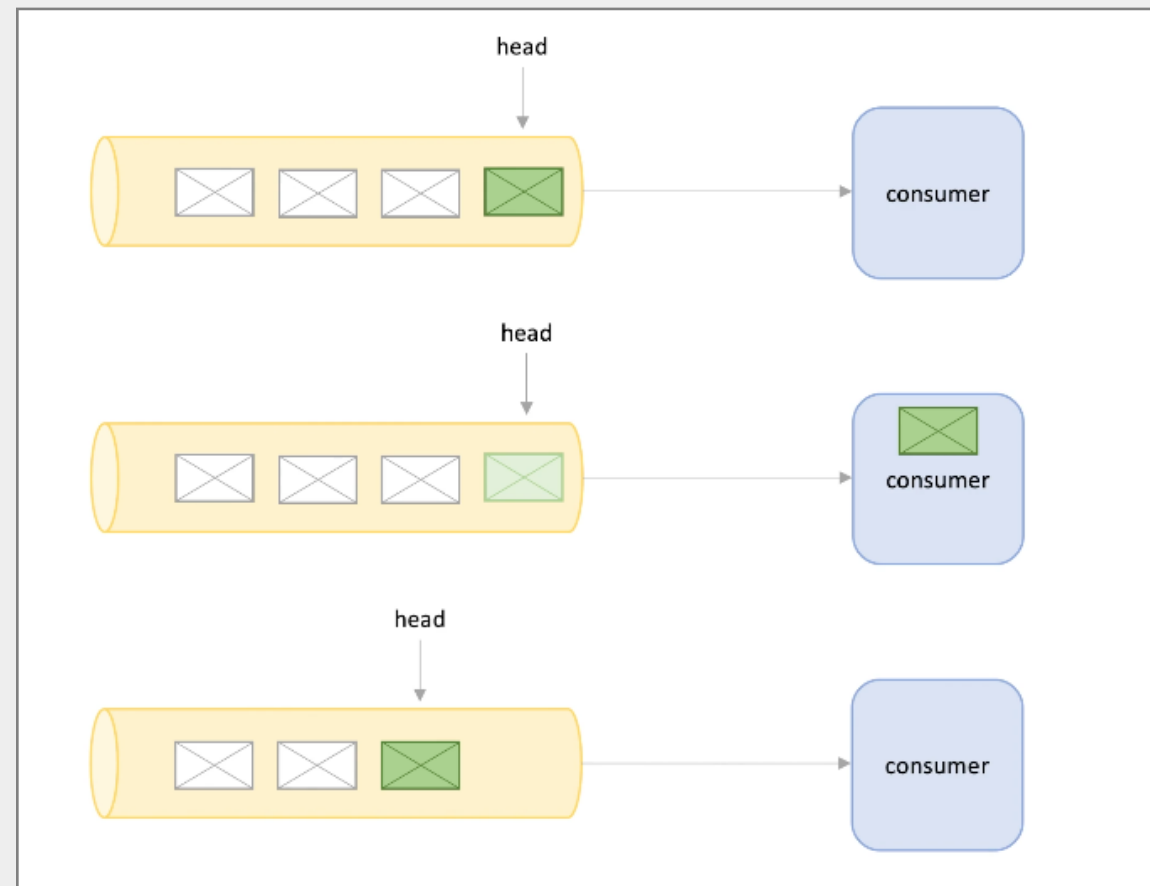


RabbitMQ, SQS

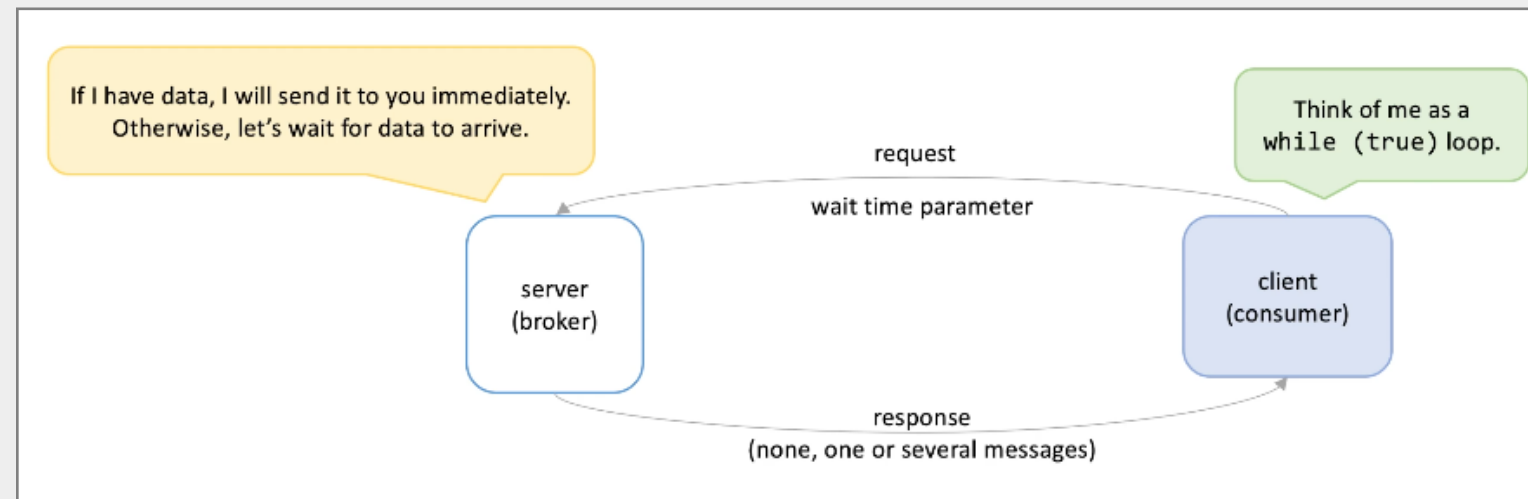[ Ack Offset Polling Guarantees ]

## Incrementing the pointer



Kafka, Kinesis

[ Ack Offset Polling Guarantees ]

## Short Polling

[ Ack Offset Polling Guarantees ]

## Long Polling

## Delivery Guarantees

`at-most-once` -> Messages might be lost

`exactly-once` -> Messages processed only one time

`at-least-once` -> Messages migth be processed multiple times

But failures will happen!

[ Ack Offset Polling Guarantees ]



" Failures are a given and everything will eventually fail over time: from routers to hard disks, from operating systems to memory units corrupting TCP packets, from transient errors to permanent failures. This is a given, whether you are using the highest quality hardware or lower cost components. "

—  Dr. Werner Vogels

Most of the messaging systems you will work with will be configured to provide `at-least-once` delivery guarantee